

Les 8 — Lesstof

Supabase CRUD + Mutations — QuickPoll voltooien

Vak: Technical Foundations
Cursus: AI Developer
Volgende les: Les 9 — Supabase Auth

Vak: Technical Foundations **Vorige les:** Les 7 — Supabase Setup **Volgende les:** Les 9 — Supabase Auth

Inhoud

1. [Waar staan we?](#1-waar-staan-we)
2. [INSERT — data toevoegen](#2-insert--data-toevoegen)
3. [De /create pagina](#3-de-create-pagina)
4. [Server actions met Supabase](#4-server-actions-met-supabase)
5. [UPDATE — bestaande rows wijzigen](#5-update--bestaande-rows-wijzigen)
6. [DELETE — rows verwijderen](#6-delete--rows-verwijderen)
7. [Voting in QuickPoll](#7-voting-in-quickpoll)
8. [TanStack Query — client-side data](#8-tanstack-query--client-side-data)
9. [Types generation](#9-types-generation)
10. [Error handling — productie-pattern](#10-error-handling--productie-pattern)
11. [QuickPoll — eindstaat](#11-quickpoll--eindstaat)

1. Waar staan we?

Vorige les (Les 7):

- Supabase project + polls tabel
- `lib/supabase.ts` met client
- Homepage leest polls vanuit DB (SELECT)

Vandaag: schrijf-operaties (INSERT, UPDATE, DELETE), de /create pagina, voting flow. Aan het eind = werkende QuickPoll, productie-klaar.

2. INSERT — data toevoegen

Basis

```
const { data, error } = await supabase
  .from("polls")
  .insert({ title: "Nieuwe poll" })
  .select()
  .single();

if (error) return { error: error.message };
console.log(data.id); // de nieuwe ID
```

`.select().single()` = krijg de ingevoegde row terug.

Meerdere tegelijk

```
await supabase.from("options").insert([
  { poll_id: 1, text: "Optie A" },
  { poll_id: 1, text: "Optie B" },
  { poll_id: 1, text: "Optie C" },
]);
```

Met returning waarden

```
const { data } = await supabase
  .from("polls")
  .insert({ title })
  .select("id, title, created_at")
  .single();
```

3. De /create pagina

Route + form

```
// app/create/page.tsx
import { createPoll } from "../actions";

export default function CreatePage() {
  return (
    <main className="max-w-md mx-auto p-8">
      <h1 className="text-2xl font-bold mb-4">Nieuwe poll</h1>
      <form action={createPoll} className="space-y-4">
        <input
          name="title"
          required
          placeholder="Welke vraag wil je stellen?"
          className="w-full border p-2 rounded"
        />
        <input
          name="options"
          required
          placeholder="Opties (comma-separated)"
          className="w-full border p-2 rounded"
        />
        <button className="bg-blue-600 text-white px-4 py-2 rounded">
          Aanmaken
        </button>
      </form>
    </main>
  );
}
```

Server action

```
// app/create/actions.ts
"use server";

import { supabase } from "@/lib/supabase";
import { redirect } from "next/navigation";
import { revalidatePath } from "next/cache";

export async function createPoll(formData: FormData) {
  const title = formData.get("title") as string;
  const optionsRaw = formData.get("options") as string;
  const options = optionsRaw.split(",").map(s => s.trim()).filter(Boolean);

  // Validate
  if (!title || options.length < 2) {
    throw new Error("Vul een titel + minstens 2 opties in");
  }

  // Insert poll
  const { data: poll, error: pollError } = await supabase
    .from("polls")
    .insert({ title })
    .select()
    .single();
  if (pollError) throw new Error(pollError.message);

  // Insert opties
  const { error: optError } = await supabase
    .from("options")
    .insert(options.map(text => ({ poll_id: poll.id, text })));
  if (optError) throw new Error(optError.message);

  revalidatePath("/");
  redirect(`/poll/${poll.id}`);
}
```

4. Server actions met Supabase

Pattern

```
"use server";
export async function someAction(formData: FormData) {
  // 1. Parse + validate
  // 2. Supabase call
  // 3. Error handling
  // 4. revalidatePath / redirect
}
```

Tips

- **"use server"** altijd bovenaan — anders runt 't in browser
- **Validate eerst** — voorkomt slechte DB rows
- **Return errors als object** ipv throw — beter voor UX
- **revalidatePath na mutation** — UI ziet anders oude data

Met return values

```
"use server";
export async function createPoll(formData: FormData) {
  // ...
  return { ok: true, pollId: poll.id };
  // of
  return { ok: false, error: "Validation failed" };
}
```

Op de client:

```
const result = await createPoll(formData);
if (!result.ok) toast.error(result.error);
```

5. UPDATE — bestaande rows wijzigen

```
const { error } = await supabase
  .from("polls")
  .update({ title: "Bijgewerkt" })
  .eq("id", 1);
```

Conditional update

```
await supabase
  .from("options")
  .update({ votes: 10 })
  .eq("id", optionId)
  .eq("poll_id", pollId); // extra safety check
```

Increment

Voor votes + 1: gebruik een **stored procedure** (RPC):

```
-- In Supabase SQL editor
create function increment_vote(option_id_input bigint)
returns void as $$
  update options set votes = votes + 1 where id = option_id_input;
$$ language sql;
```

```
await supabase.rpc("increment_vote", { option_id_input: 5 });
```

6. DELETE — rows verwijderen

```
const { error } = await supabase
  .from("polls")
  .delete()
  .eq("id", 1);
```

Met on `delete cascade` (uit Les 7) verdwijnen options automatisch mee.

Belangrijk

`.delete()` **zonder** `.eq(...)` verwijdert ALLES. Altijd een filter erbij:

```
// ■ NIET – leeg de hele tabel
await supabase.from("polls").delete();

// ✓ wel
await supabase.from("polls").delete().eq("id", 1);
```

7. Voting in QuickPoll

Voting form

```
// components/VoteForm.tsx
import { vote } from "../actions";

export function VoteForm({ pollId, options }) {
  return (
    <form action={vote} className="space-y-3">
      <input type="hidden" name="pollId" value={pollId} />
      {options.map(o => (
        <label key={o.id} className="flex items-center gap-2">
          <input type="radio" name="optionId" value={o.id} required />
          <span>{o.text}</span>
          <span className="ml-auto text-sm text-gray-500">{o.votes} stemmen</span>
        </label>
      ))}
      <button className="bg-blue-600 text-white px-4 py-2 rounded">Stem</button>
    </form>
  );
}
```

Vote action

```
"use server";
export async function vote(formData: FormData) {
  const pollId = Number(formData.get("pollId"));
  const optionId = Number(formData.get("optionId"));

  await supabase.rpc("increment_vote", { option_id_input: optionId });
  revalidatePath(`/poll/${pollId}`);
}
```

8. TanStack Query — client-side data

Voor interactieve UI's (live updates, optimistic state) is useEffect + fetch matig. **TanStack Query** is de standaard.

Install

```
pnpm add @tanstack/react-query
```

Setup

```
// app/providers.tsx
"use client";
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
import { useState } from "react";

export function Providers({ children }) {
  const [qc] = useState(() => new QueryClient());
  return <QueryClientProvider client={qc}>{children}</QueryClientProvider>;
}
```

```
// app/layout.tsx
import { Providers } from "../providers";

export default function RootLayout({ children }) {
  return <html><body><Providers>{children}</Providers></body></html>;
}
```

Gebruik

```
"use client";
import { useQuery } from "@tanstack/react-query";

export function LivePollResults({ pollId }) {
  const { data, isLoading } = useQuery({
    queryKey: ["poll", pollId],
    queryFn: () => fetch(`/api/polls/${pollId}`).then(r => r.json()),
    refetchInterval: 2000, // poll elke 2 sec
  });
  if (isLoading) return <p>Loading...</p>;
  return <PollChart options={data.options} />;
}
```

Voor mutations: `useMutation`. Voor cache-invalidation: `queryClient.invalidateQueries`.

9. Types generation

Supabase kan TypeScript types genereren vanuit je schema. Geen handmatig typen meer.

Install CLI

```
pnpm add -D supabase
pnpm dlx supabase login
pnpm dlx supabase link --project-ref <project-id>
```

Genereer


```
pnpm dlx supabase gen types typescript --linked > types/database.ts
```

Gebruik

```
import { createClient } from "@supabase/supabase-js";
import type { Database } from "@types/database";

export const supabase = createClient<Database>(...);

// Nu krijg je autocomplete + type-safe queries
const { data } = await supabase.from("polls").select("title");
// data is typed: { title: string }[] | null
```

Herhaal na schema-wijzigingen. Of automatiseer via CI.

10. Error handling — productie-pattern

Wrap Supabase calls

```
async function safeSupabaseCall<T>(  
  fn: () => Promise<{ data: T | null; error: Error | null }>  
) : Promise<{ ok: true; data: T } | { ok: false; error: string }> {  
  try {  
    const { data, error } = await fn();  
    if (error) return { ok: false, error: error.message };  
    if (data === null) return { ok: false, error: "No data" };  
    return { ok: true, data };  
  } catch (e: any) {  
    return { ok: false, error: e.message };  
  }  
}
```

Gebruik

```
const result = await safeSupabaseCall(() =>  
  supabase.from("polls").select("*").single()  
);  
if (!result.ok) return <Error message={result.error} />;  
return <Poll poll={result.data} />;
```

11. QuickPoll — eindstaat

Wat draait er nu:

-
- **Homepage** — lijst polls (server-side fetch)
 - **/create** — nieuwe poll form (server action)
 - **/poll/[id]** — detail + voting (server action voor vote)
 - **Database** — polls + options tabellen met RLS open policies
 - **Types** — gegenereerd vanuit Supabase

Wat ontbreekt nog

- **Auth** — iedereen kan polls maken/stemmen (komt in Les 9)
- **RLS per user** — users alleen hun eigen polls (komt in Les 10)
- **Realtime** — votes updaten live zonder refresh (later)
- **Polish** — chart-visualisaties, deel-links (later)

Maar: **QuickPoll werkt end-to-end op een echte database**. Grote stap.

Bronnen

- **Supabase JS — insert:** <https://supabase.com/docs/reference/javascript/insert>
- **Supabase JS — update:** <https://supabase.com/docs/reference/javascript/update>
- **Supabase JS — delete:** <https://supabase.com/docs/reference/javascript/delete>
- **Supabase RPC:** <https://supabase.com/docs/reference/javascript/rpc>
- **TanStack Query:** <https://tanstack.com/query/latest/docs/framework/react/overview>
- **Type generation:** <https://supabase.com/docs/guides/api/rest/generating-types>
- **Next.js Server Actions:** <https://nextjs.org/docs/app/getting-started/updating-data>